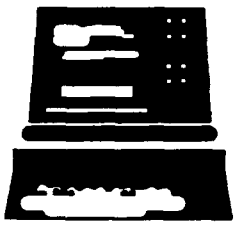


USAISEC

US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300

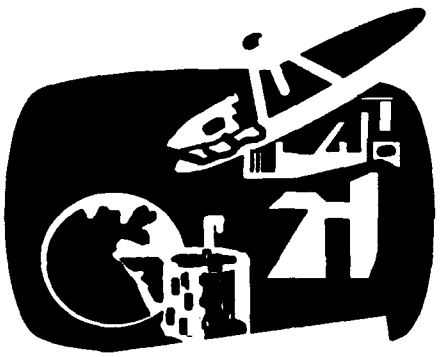
U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

INTEGRATION OF SEVERAL COMPUTER SYSTEMS WITHIN A HETEROGENEOUS ENVIRONMENT

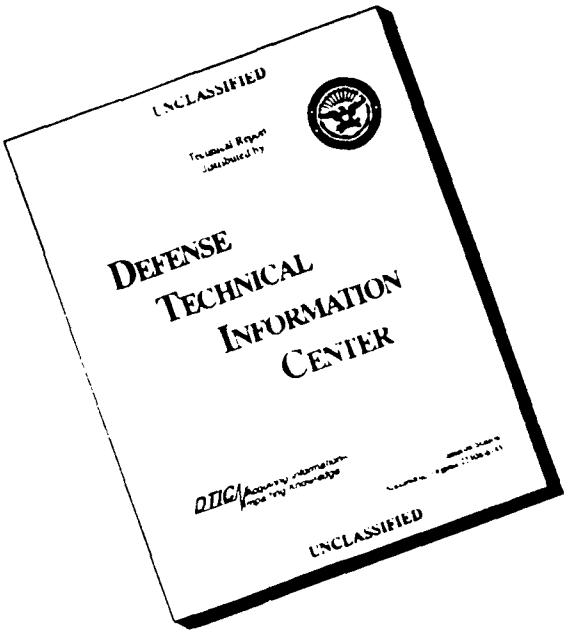
ASQB-GM-92-024

September 1992

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800



DISCLAIMER NOTICE



THIS REPORT IS INCOMPLETE BUT IS THE BEST AVAILABLE COPY FURNISHED TO THE CENTER. THERE ARE MULTIPLE MISSING PAGES. ALL ATTEMPTS TO DATE TO OBTAIN THE MISSING PAGES HAVE BEEN UNSUCCESSFUL.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704--188
Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION / AVAILABILITY OF REPORT N/A
2b. DECLASSIFICATION / DOUWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ASQB-GM-92-024	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A
6a. NAME OF PERFORMING ORGANIZATION AIRMICS 6b. OFFICE SYMBOL (if applicable) ASQB - GM	7a. NAME OF MONITORING ORGANIZATION N/A
6c. ADDRESS (City, State, and ZIP Code) 115 O'Keefe Bldg., Georgia Institute of Technology Atlanta, GA 30332-0800	7b. ADDRESS (City, State, and Zip Code) N/A
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AIRMICS 8b. OFFICE SYMBOL (if applicable) ASQB - GM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 115 O'Keefe Bldg., Georgia Institute of Technology Atlanta, GA 30332-0800	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: 62783A PROJECT NO.: DY10 TASK NO.: 00-08 WORK UNIT ACCESSION NO.:

11. TITLE (Include Security Classification)

Integration of Several Computer Systems Within a Heterogeneous Environment (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)

William O. Putnam, Ian E. Smith

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM _____ TO _____	September 1992	52

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Information System Architecture, Information Architecture Reference Model (IARM)

19. ABSTRACT (Continue on reverse if necessary and identity by block number)

This project prepared the AIRMICS testbed for an evaluation of the Information Architecture Reference Model (IARM). Plans are to perform the evaluation by integrating a set of independently developed applications into a heterogeneous environment.

The project also modified and enhanced the IARM to incorporate it into the Army's Information Systems Architecture Circa 1997 plan (ISA97).

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED / UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Gerard C. McCoyd	22b. TELEPHONE (Include Area Code) (404) 894-3110 22c OFFICE SYMBOL ASQB - GM

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Executive Summary

This report describes a project which prepared the AIRMICS testbed for an evaluation of the Information Architecture Reference Model (IARM). Plans are to perform the evaluation by integrating a set of independently developed applications into a heterogeneous environment. The project also modified and enhanced the IARM to incorporate it into the Army's Information Systems Architecture Circa 1997 plan (ISA97).

Contents

SPARC workstations in the testbed. The task of obtaining and installing the X Window software will be performed under this project. Plans for enhancement of the testbed network will be developed.

Study of the IARM continued. The model required changes to accommodate the Army's Information Systems Architecture Circa 1997. A need for more information on open systems standards and technologies was identified.

During the month of March the X Window System software was obtained and loaded onto a Sun 3 workstation in the AIRMICS network. The system is quite large (over 150MB of source code) and required extensive study to configure and build.

Work was also done on the development of the IARM and its integration with the ISA97 plan. We decided to merge the two into a single model of information system architecture. Information on open systems standards and technologies was gathered.

AIRMICS PC systems were integrated into the network using PC-NFS software for TCP/IP communications. Supported applications include remote printing, file transfer, electronic mail, and remote file systems mounted from Sun workstations and the AIRMICS file server. We began examining interface compatibility products to provide a consistent user interface between DOS and UNIX systems.

During the month of April the primary focus was on the installation of the X Window System on the Sun 3 workstations. The software was loaded, configured, compiled, and tested. Once Sun 3 installation was complete the software had to be re-compiled for the 386i workstations. We also investigated X products for the DOS systems in the network.

The focus of the project shifted from integration of ANSWER, RAID, and IOIS to refinement of the IARM into the ISA 97 Compliant Architecture Model (ICAM). The ICAM will describe the components of an information system and relate them to open systems standards and technologies. It was decided that this project will focus on the definition of the Entry and Operating System layers of the model.

During the period from May 1 to May 31, 1990 several steps were taken to continue the upgrading of the AIRMICS research network. Information was collected on the costs and capabilities of several commercial hardware and software products of interest to AIRMICS including X servers for PC systems, X terminals, and diskless/dataless SPARC workstations. A report describing the information collected and considering the pros and cons of the technologies was prepared and submitted. Also, software to control the uninterruptible power supply on the AIRMICS file server was installed. A special cable was required to connect the UPS control port to the server due to the incomplete implementation of the RS232 standard on the server's ALM serial card unit. Installation of the X window system on the Sun 3 systems was completed and work began on the 386i systems. SunOS 4.1 for the Sun 3 systems arrived but could be installed until memory upgrades for the 3/50 workstations were installed.

During the period from June 1 through June 30, 1990, efforts were focused on the development of an ISA 97 architecture model and proof of concept testbed. The objectives were:

ponent functions and these functions described. The expanded material was formatted with Interleaf and placed on the AIRMICS file server for inclusion with other ICAM briefing material.

Two reports on the portability of GUI-based applications from proprietary environments to the X Window System were prepared. The reports are based on the experience gained from developing the WYWO application first in the SunView environment and then porting it to X using both the XView portability package and toolkit and the MIT X toolkit. These reports are included here as Appendices 3 and 4.

The FIPS publications for POSIX (151) and GOSIP (146) arrived and were studied. We hope that these publications will give us a basic understanding of the standards and will lead us to other documentation. Our goal is to develop a transition strategy or strategies for the migration to open systems.

During the month of October discussion of the ICAM model continued at weekly meetings held at AIRMICS. The Entry and Operating Systems modules were given further attention, with the intent of defining services provided by those modules to the Application module.

The Sun workstation equipment arrived and was installed in the AIRMICS network. The process of moving user accounts to the new machines, installing Interleaf and other applications software, and converting data files to the SPARC format was quite time consuming. An automated script developed for the Interleaf data conversion was helpful.

Backup of the AIRMICS file server became a problem due to the number of 1/2 inch tapes required. The process was taking 10 tapes and most of a day to perform, making the system unavailable for substantial periods. We began looking into the possibility of getting an 8 millimeter Exabyte tape drive which would back up the entire system on a single 2.5GB tape unattended.

We began gathering information on COBOL compilers for the Sun SPARC systems to use in porting STAMMIS applications into the testbed environment. We also began looking at DOS emulator software for SPARC systems and UNIX work-alike software for the DOS systems. This software would give a common user environment at the command interpreter level on both types of systems.

During the month of November the integration of the new Sun SPARCstations into the AIRMICS network was completed. User accounts are now accessible on all Sun systems and files are mounted from the AIRMICS server using NFS.

Refinement of the ICAM Entry module services continued and created interest in GUI standards and development environments. The OpenLook GUI standard is implemented on the Sun SPARCstations and is operational in the AIRMICS testbed. The MIT X Window System which was installed on the Sun 3 equipment can interoperate with OpenLook, but is not integrated – it has a different "look and feel". We wished to standardize the look and feel of the GUI across all systems. This was to be addressed in three ways: 1) install the MIT X release on the SPARCs as well as the Sun 3 systems; 2) obtain OpenLook for the Sun 3 systems; 3) obtain the Motif GUI standard software (the competitor to OpenLook) and in-

3.0 Conclusions

With the installation of the X window system software in the AIRMICS testbed we are now ready to install and integrate ANSWER and RAID. Other open system technologies such as Graphical User Interface development tools, database integration tools, and networking tools have been discovered during our migration and should be examined more closely. We believe that several of these tools may be applicable to the problem of transitioning Army information systems from the proprietary mainframe environment into the distributed open systems computing environment.

Of particular interest are CASE and reverse engineering tools such as IDE Software Through Pictures, GUI development tools such as ICS Builder Xcessory, graphical shell and migration tools such as IXI Deskterm, and PC X software such as PC/Xview. A vendor survey should be conducted and selected products obtained for evaluation in the testbed.

It should be possible to migrate a selected STAMMIS into the testbed using these technologies to enhance the user interface and integrate the application with a relational database system.

The IARM has evolved into the ISA97 Compliant Architecture Model (ICAM). Further refinement of all six ICAM modules is needed. Some detail has been provided for the User Interface and Operating System modules. Slides detailing these modules were developed for an ICAM briefing and are provided in Appendix 1.

The AIRMICS IARM testbed has evolved into the ICAM Testbed (ICAT) network. All workstations of the testbed now support the X Window System and the Motif and OpenLook user interfaces. Further evaluation of these two competing interface technologies is needed. It is not yet clear which will prevail in the commercial arena.

Using the enhanced testbed, it should now be possible to implement a demonstration of the ICAM using selected applications and networking technologies. One area of concern is networking: how will GOSIP affect the Operating System and User Interface modules? The X Window system is TCP/IP based at this time, but there are indications that a migration to GOSIP is being considered [see "Mapping the X Window onto Open Systems Interconnection Standards"; Brennan, Thompson, & Wilder; IEEE Network Magazine, May 1991].

4.1 Appendix 1

ISA 97 Briefing Slides
Operating System and User Interface Modules

William Putnam
College of Computing
Georgia Institute of Technology

This slide set was prepared for use in a technical briefing on the ISA 97 Architecture Model and Testbed.

USAISEC

ISA 97 Conceptual Architecture

User to Application Interfaces and Standards

The user's view of the application is shaped by the entry module. At this level the nature of the interaction between user and application is defined. Input and output devices, such as displays, keyboards, and pointing devices, are provided and standards for their operation are defined. The user sees the devices and the rules for their operation.

The application developer sees the details of the interaction between the devices and the application software. The developer uses toolkits to build support for the devices into the application. These toolkits contain the implementation details of standard components of the user interface, *such as menus, scrollbars, buttons, and so on.*

AIRMICS

USAISEC

ISA 97 Conceptual Architecture

User to Application Interfaces and Standards

This slide presents definitions of the primary concepts of the User Interface and Application Standards.

AIRMICS

USAISEC

ISA 97 Conceptual Architecture

User to Application Interfaces and Standards

In this view of the User-to-Application modules we show some of the existing standards and implementations of the interfaces.

The user has available a selection of input and output devices including monochrome and color displays, keyboards, mice, touchpads, trackballs, and so on.The operation of these devices is under the control of the Entry module, which implements some user interface model. The most common model today is the Graphical User Interface (GUI), which usually provides a workspace ("the desktop") containing logically distinct regions ("windows") for applications to display information and accept input. The workspace and its components are under the control of a process which is separate from the applications. Applications must cooperate with this process via standard interfaces to use system display and input resources.

The X Window system is a widely accepted GUI for many hardware platforms. It uses the desktop and window metaphor and is based on the client-server model. The "X server" manages the physical display resources. Applications are "clients" of the server and make requests to use resources.

The Application Program Interface (API) between the application and the X server consists of programming libraries called "toolkits". These toolkits supply the basic building blocks of the user interface: the windows, menus, scrollbars, buttons, etc.

The other component of the Entry module is the "look and feel" of the interface. This is defined by the arrangement of the windows, the assignment of functions to buttons, the style and operation of menus, and so on. These elements are specified in documents called style guides or interface specifications. Some of the rules and constraints in the style guide are built into the toolkits used by the application developer. Others are left to the display server or some independent process for implementation and enforcement.

In the X Window system most of these details are implemented by the "window manager", a process which coordinates the fine detail of the GUI operation, leaving the X server free to handle the raw resources. This separation of function allows users great freedom to customize their computing environments to suit their individual tastes and needs. The X system is independent of any look and feel. The applications can be made independent also. The look and feel component of the GUI is then easily changed or replaced without affecting the majority of the system.

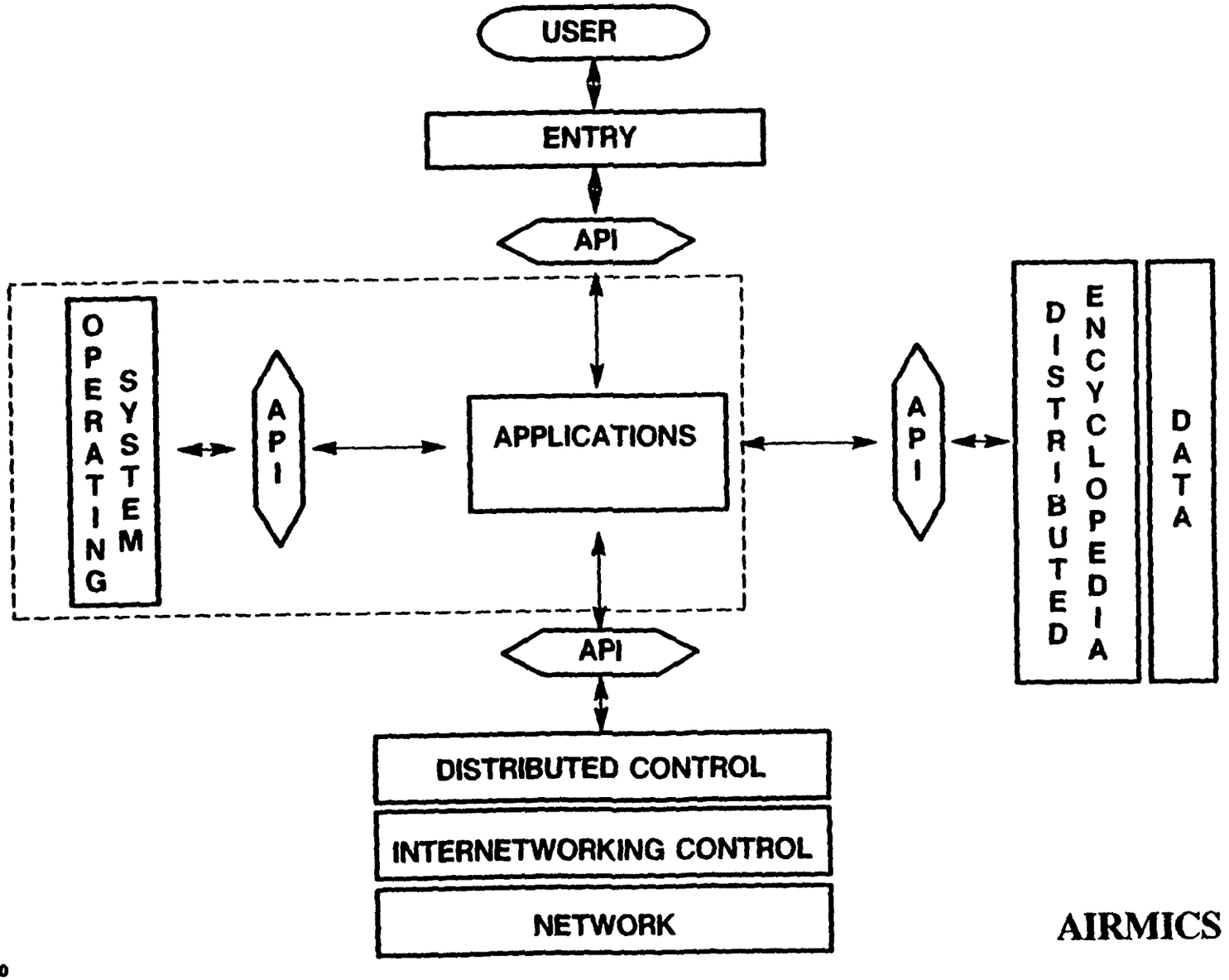
Some examples of style guides, window managers, X toolkits, and typical application programs are shown above.

AIRMICS

USAISEC

ISA 97 Conceptual Architecture

Operating System to Application Interfaces and Standards



AIRMICS

USAISEC

ISA 97 Conceptual Architecture

Operating System to Application Interfaces and Standards

POSIX – specify standards for characteristics of operating system environment, including system resources and management, job & process control, portability & interoperability standards

Libraries and Toolkits – provide application developers with access to system functions and resources through standardized components at various levels.

Applications – may be DBMS, X servers & clients, networking programs & tools, or command shells for users. Applications must conform to the operating system standards for resource allocation, execution, and communications.

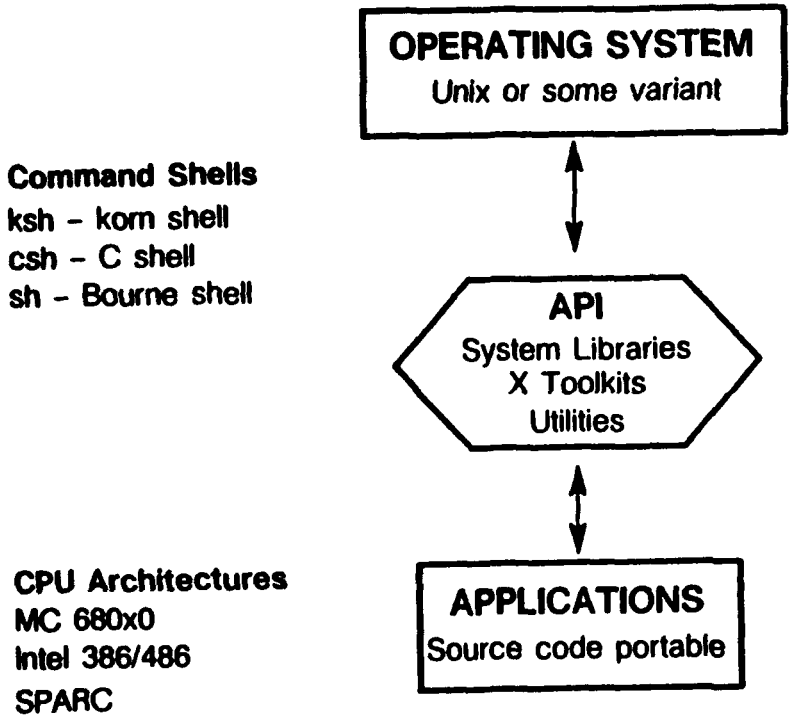
Utilities – programs used to maintain system resources

AIRMICS

USAISEC

ISA 97 Conceptual Architecture

Operating System to Application Interfaces and Standards



Command Shells
ksh - korn shell
csh - C shell
sh - Bourne shell

CPU Architectures
MC 680x0
Intel 386/486
SPARC
MIPS

POSIX implementations

SunOS 4.1	Ultrix	Mach
OSF/1	4.3BSD	SysV.4

Libraries

math	network access & management
windowing	*file & data management*

Utilities
account management
user registration
backup & recovery

From the Operating System perspective, *everything* looks like an application.

AIRMICS

USAISEC

ISA 97 Conceptual Architecture

Operating System to Application Interfaces and Standards

The desire for common operating system across all hardware platforms lead to the development of the POSIX operating system standard. POSIX specifies the methods of resource allocation and management, interprocess communication, and other details of system operation. It specifies the basic utilities to be included in the system for maintenance and operation. These are implemented by standard system function libraries and utility programs.

Many of the concepts of the unix operating system are present in the POSIX standard, and many of the available unix implementations are or soon will be POSIX compliant. Some of the POSIX compliant systems and their components are shown above. Command shells provide a direct user interface to the operating system. Libraries are used by application developers. Utility programs are used to maintain system resources.

POSIX is designed to run on many different hardware architectures, including those shown above.

AIRMICS

USAISEC

ISA 97 Conceptual Architecture

Development and Evaluation Testbed

This slide shows the ISA 97 Architecture Development and Evaluation Testbed being developed at AIRMICS.

The testbed demonstrates the concepts described above in a heterogeneous, distributed, multiply connected network environment. Its features include a portable distributed GUI based on the X Window system, three different hardware architectures (Intel 80x86, Motorola 680x0, RISC), GOSIP-compliant local and wide area networking with ISDN and Ethernet, and POSIX-compliant operating systems.

AIRMICS

4.3 Appendix 3

Converting Sunview Applications to X

An Overview

Ian Smith
College of Computing
Georgia Institute of Technology

Introduction

This document will discuss the problems involved in converting an existing Sunview based application to X. The challenges involved in porting an application are many and varied. They range from simple correspondences to complex design issues. In this paper we will discuss several of the major issues that are evinced in this process. Principal among these problems is the one of "enforcement." X gives the programmer and user all the freedom that they want (or may not want); Sunview enforces user interface rules. To compound this problem, there are areas of Sunview that simply do not correspond to any simple type of construction in X. There is also the problem of getting high quality documentation. All of these issues will be discussed, as well as potential solutions.

Background

Before one can understand the problems in porting programs from Sunview to X, a bit of background is needed. The most significant difference in the way X applications are written versus Sunview is that X has the concept of 'Widgets.' These widgets are user interface construction blocks -- like scrollbars, buttons, and windows -- which the application programmer uses to construct the application. Sunview does not have this concept. Several widgets are usually put together into 'Widget Sets' (sometimes called 'toolkits') in which all widgets function together and use similar user interface conventions. Many vendors market widget sets and several others are available for free. Some of the more popular widget sets include (with authoring organization in parenthesis) the Athena Widgets (MIT), the HP Widgets (Hewlett Packard), the XView toolkit (Sun), the Andrew toolkit (CMU) , and the Motif Widget set (OSF). Of these, only Motif is a 'for-sale' product.

There are many widget sets available, each with their own strengths, weaknesses, and user-interface. This can be a problem if the user is confronted with several programs that function differently. This problem does not exist in Sunview. In this manner, Sunview can be thought of a window system that has exactly one widget set. We will see later that widget sets provide both a problem and a potential solution in our attempts to convert Sunview applications.

Sunview Enforces Policy -- X Does Not

Most of the difficulty involved in porting an application from Sunview to X revolves around, if not hinges upon, the fact that X does NOT enforce any type of user interface (UI) policy

The XView toolkit is a set of programs written by Sun in an attempt to make X programming feel like Sunview programming. The XView user interface is different than the Sunview one. XView however is not a panacea. As mentioned before, X and Sunview are considerably different and XView suffers when it tries to make X an exact match of Sunview. To its credit, XView is considerably easier to program in than most widget sets, and does in fact make X programming 'feel' like Sunview. However, its problems are considerable. I was unable in some cases to get XView to reproduce behavior of a Sunview program. The XView documentation, unlike the Sunview documentation, is poor. Additionally, XView is Open Look compliant, and it expects an OL compliant window manager to function perfectly. This is not a major drawback, but can be annoying.

Sun provides with XView a set of scripts that are supposed to convert some or all the of the Sunview code to XView. As far as I could tell these were of little value. These scripts generated XView code correctly, but what they generated differed only slightly from the Sunview code. Also, because they were automated, they certainly did not provide any assistance in areas of the code that were proving difficult to port. Converting the code by hand, with the XView conversion documentation proved at least as easy as the automated process. Additionally, this procedure allowed the programmer to use his own intelligence and knowledge of the code to produce higher quality output.

The Athena Widgets are being considered in this paper more as a representative of the normal type of widget set than anything else. Normal widget sets are those that obey the normal X conventions for widget definition and usage, which XView does not. XView follows the Sunview conventions. Porting an Sunview application to X with the Athena Widgets is difficult too. In many cases, the code has to be revamped in order to fit completely within the Athena Widgets constraints. However, due to the fact that the athena widgets comply with the X standards for widget sets, lower level calls can be invoked (if the programmer desires) so that no portion of the interface has to change from a user perspective. This basically allows the programmer to easily subvert the constraints of the widget set.

It should also be noted that much of the third party documentation on X (in fact nearly all of it) expects 'normal' widget sets, like the Athena Widgets. There recently has been a book published, however, on XView programming. I have not reviewed this book.

Conclusion

In conclusion the porting of programs from Sunview to X suffers from three major stumbling blocks:

1) Areas that do not map easily from Sunview to X;

2) Making the correct choice for the widget set in the X environment;

3) Inadequate documentation.

Of these, the second and third can be avoided almost completely if sufficient time and energy are spent prior to the commencement of the project. The first problem is more difficult. In most cases, and experienced X and Unix programmer can work around the

4.4 Appendix 4

Converting Sunview Applications To X

Some Notes for Programmers

Ian Smith
College of Computing
Georgia Institute of Technology

Introduction

This document will describe the problems involved in porting Sunview applications to X. It is assumed that the reader has a general knowledge of unix, X, and Sunview. It will cover specific problem areas in some detail, and will probably only be of interest to those actually doing ports. It assumes that reader has a system running unix (bsd 4.2 or greater), X11R4, and Sunview. This document should be considered as a guideline only, and your port should dictate its own individual needs.

How to write Sunview programs that are easy to port

It cannot be stressed enough how much good, original design will speed the porting process. The most import thing to keep in mind when writing a Sunview program that might be ported to X is separation. Separate the user interface from the functionality as much as possible. The following separation guidelines can be helpful when Sunview work is done.

1) Physically separate the user interface (front end) from the functionality of the program. This can be accomplished with separate files and/or directories. If you are careful to do this porting time can be reduced by large amounts.

2) Write the functionality (back end) as unix program that uses stdin and stdout then write the front end for it. You can usually accomplish this easily via the system call popen. If you require more sophisticated interfaces you can use the fork/exec/pipe sequence of system calls.

3) Document the user interface as thoroughly as the functionality. Good documentation, especially of global variables that proliferate in window system applications, can save a lot of time later on.

How to use X resources efficiently in a porting situation

In general, make all available use of the X resource database. During the process of porting the application, you should generally try to map all "attribute-value" pairs in the Sunview code to an X resource. When you converting the code, as always, try to avoid putting any constants in the X source. This accomplishes two things: First, it makes the program amenable to user customization. Second, it speeds up the porting process by avoiding the need to compile potentially large amounts of unmodified code.

In particular, pay special attention to the translation manager and its translation tables. Good use of these tables in the resource file can give you emulation of the Sunview user

4) Use XSetIconName(...) to hint to the window manager what you would like your icon's name to be.

It should be noted that if the window manager does not use the window property WM_HINTS for communicating with clients, this method is useless.

Cursors

Cursors are similar in X and Sunview. The only significant difference to be aware of is Sunview prefers you to define your cursor at the time a window is created and X does not. In X you use XDefineCursor to associate a cursor with a window.

There is a trade off associated with using the standard X cursors. If you use one of the predefined cursors (there are about 75 cursors distributed with X) in a manner similar to the way other applications use it, you will have an important clue for X users in how your application is operating. Conversely, if you create a custom cursor that is similar to your Sunview cursor you will have a smaller learning curve for users familiar with the Sunview version. These two sides must be weighed on an individual bases for every port.

Focus (popups and warnings)

The X focusing model is less restrictive than Sunview, in general. If your code uses the Sunview split focus model, be sure that you use the grab functions (XGrabPointer, XGrab-Keyboard,XGrabServer, and most important of all for widget programmers XtAddGrab) to control explicitly which window is getting user input.

In the most part you are going to want to force focus into a widget that has some type of urgent message or warning. If you use the athena dialog widget you can get the widget set to do this for you. Otherwise you will need to call XtAddGrab with appropriate parameters, thus forcing the user to deal with your warning message.

Constraint widgets

In Sunview there are basically two "constraint" widgets, the frame and the panel. The Athena analogues of those two widgets are the form and the paned widget. These are the two basic constraint widgets in the Xaw library and have similar functionality to their Sunview counterparts. (Note: Most other widget sets have similarly named widgets with the same functionality.) However, often X composite widgets can serve needs that are met be complex constructions under Sunview. Be sure to look carefully at the Viewport widget, which is often used to pan a window over a large virtual surface.

Do not feel limited by the constraint widgets of the Athena widgets. Many other widget sets have constraint widgets that will work successfully Athena widget children. It is often quicker to look for a constraint widget that implements the layout semantics you want than to use a widget that does not easily support your layout. The HP widgets are an especially good source of constraint widgets.

Look and feel

Look and feel can be difficult point when porting Sunview applications. In general, it most important to preserve the functionality first, then make a functioning program look and feel

4.5 Appendix 5

An Introduction to the X Window System

Ian Smith
College of Computing
Georgia Institute of Technology

This slide set was prepared for use in an introductory briefing on the MIT X Window system.

USAISEC

History

- X was originally developed at M.I.T. to solve the problems of heterogenous networks of bitmapped workstations.

- The first public release of X was in 1986 with version 10R4.

- The current release of X is 11R4 with R5 expected out in 1991.

AIRMICS

USAISEC

X and Open Systems

- X is an 'open system'. The X protocol is free and easily available.
- Anyone can modify X to suit their particular needs.
- The X sample servers are not 'supported' like a commercial product.

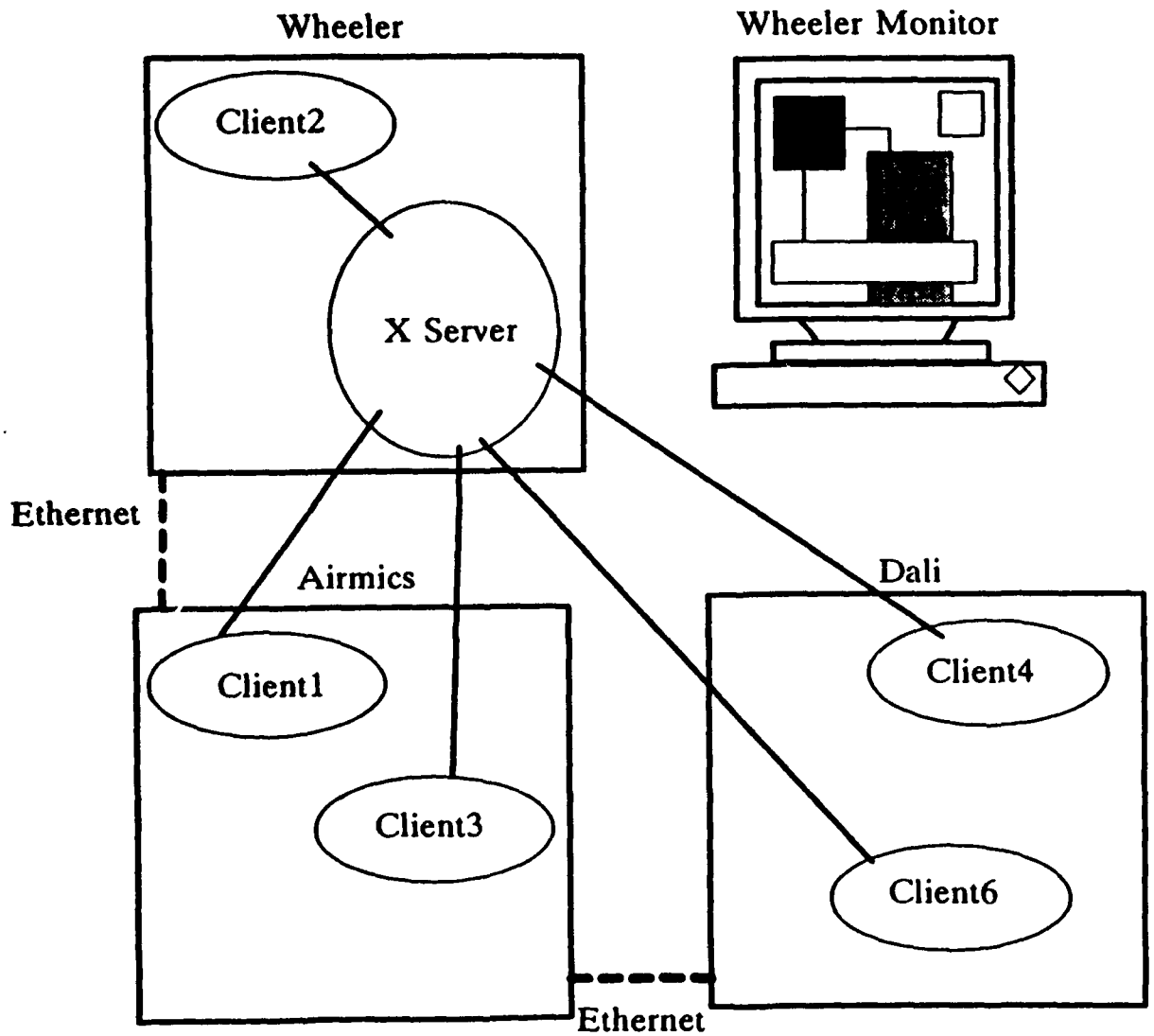
AIRMICS

USAISEC

Why You Might *Not* Want To Use X

- X is a large system, and has a substantial learning curve.
- X sample servers are not supported by M.I.T.
- Raw X expects user to understand significant amounts of information about the window system itself to reap the benifits of the system.

AIRMICS



Conceptual X Model

USAISEC

Setting Up X On Your Workstation

- Many things in X are user customizable. To install a default configuration on your workstation use:

  % xsetup

- To start X on your workstation use:

  % startx

- To quit X on your workstation select 'quit' on the pull-down menu of the background.

AIRMICS

USAISEC

X Security

- The sample X server's granularity of security is coarse.

- A primitive form of security is provided by the program xhost. You use xhost to tell your server which machines may access your display. I.e.,

  % xhost +wheeler

AIRMICS

USAISEC

If Something Goes Wrong...

- If the program will not run and says "Can't Open Display" check to make sure your DISPLAY environment is correct and that you have set your xhost's properly.

- If an application runs but seems to look strange (fonts are too small, colors are weird, etc.) then check to make sure that the programs "resources" are loaded. You load resources with the program 'xrdb.'

AIRMICS

USAISEC

Where X Is Headed

- Mixed media 'displays' with audio and real–time video.
- More commercial support.
- As X gets more of a share of the end–user market, there more 'insulation' available to the user from the complexities of the system.

AIRMICS